盤面の「良さ」を数値化する

Phase2

このPhaseの目標

貪欲法は目の前の利得を最大化しますが、その後の展開を考慮しません。このフェーズでは、**評価関数 (Evaluation Function)** を導入し、現在の盤面の状態がAIにとって「どれだけ有利か」を数値化します。

この評価関数は、次のフェーズ（探索）の土台となります

- Board.javaに盤面を複製し、着手を適用するメソッドを追加する。
- ThinkingEngine.javaにevaluateBoardメソッドを追加し、角や辺の概念を導入して盤面評価を向上させる。
- 重み付けを調整し、AIの戦略的な挙動が変化するのを確認する。

評価関数の設計

評価関数は、単なる石の数の差（自分の石数 - 相手の石数）よりも、以下の戦略的な要素を重視する必要があります。

評価項目	重要度	戦略的な意味
石の数差	低	単なる最終結果。序盤・中盤では重要度が低い。
角	最高	一度とると絶対に裏返らないため、非常に有利。
辺	中	辺も裏返りにくいが、角を取られると脆弱になる。
着手可能数	高	自分の打てる手が多いほど有利

2.貪欲法の改善

Phase1で実装した貪欲法を、評価関数を利用するように変更します。

- 変更前：裏返せる石の数が最も多い手を選ぶ。
- 変更後：着手を打った**直後の盤面**を評価関数に渡し、**最も評価値が高い盤面**になる手を選ぶ。

ヒント：think()の修正

```
    /**
     * 現在の盤面情報に基づいて最適な着手を決定する。
     * @param currentBoard 現在の盤面オブジェクト
     * @return 着手文字列（例: "c5" または "pass"）
     */
    public String think(Board currentBoard) {
        int myColor = currentBoard.getMyColor();
        List<String> legalMoves = currentBoard.getLegalMoves(myColor);
        if (legalMoves.isEmpty()) return "pass";

        String bestMove = legalMoves.get(0); // 仮の最善手
        int maxScore = Integer.MAX_VALUE;
        for (String move : legalMoves) {
            int score = evaluateBoard(currentBoard, move, myColor); // 評価する
            if (maxScore < score) {
                maxScore = score;
                bestMove = move;
            }
        }
        return bestMove; // 最も裏返し数が少ない手
    }
```

ヒント：evaluateBoard()

```
private int evaluateBoard(Board board) {
    int myColor = board.getMyColor();
    int myScore = board.countStones(myColor);                    // 自分の石の数
    int oppScore = board.countStones(board.getOpponentColor()); // 相手の石の数

    // 1. 角の重み付けを実装
    int cornerWeight = 100;
    int myCorner = 0;

    // 角の座標 (0,0), (0,7), (7,0), (7,7) をチェック
    if (board.getCell(0, 0) == myColor) myCorner++;
    // ... 他の角もチェック ...

    // 辺の評価
    int edgeWeight = 30;
    int myEdge = 0;
    // todo: 辺を数える

    // 着手可能数の評価
    int possibleWeight = 70;
    int myPossible = 0;
    // todo: 着手可能数をセットする


    // 2. 評価値の計算
    //
    int evaluation = (myScore - oppScore) * 1 + myCorner * cornerWeight + myEdge * edgeWeight + myPossible * possibleWeight;

    return evaluation;
}
```